John von Neumann Institute for Computing

Direct Solvers for Symmetric Eigenvalue Problems

Bruno Lang

published in

Modern Methods and Algorithms of Quantum Chemistry,
Proceedings, Second Edition, J. Grotendorst (Ed.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **3**, ISBN 3-00-005834-6, pp. 231-259, 2000.

http://www.fz-juelich.de/nic-series/

DIRECT SOLVERS FOR SYMMETRIC EIGENVALUE PROBLEMS

BRUNO LANG
Aachen University of Technology
Computing Center
Seffenter Weg 23, 52074 Aachen, Germany
E-mail: lang@rz.rwth-aachen.de

This article reviews classical and recent direct methods for computing eigenvalues and eigenvectors of symmetric full or banded matrices. The ideas underlying the methods are presented, and the properties of the algorithms with respect to accuracy and performance are discussed. Finally, pointers to relevant software are given.

This article reviews classical, as well as recent state-of-the-art, direct solvers for standard and generalized symmetric eigenvalue problems. In Section 1 we explain what direct solvers for symmetric eigenvalue problems are. Section 2 describes what we may reasonably expect from an eigenvalue solver in terms of accuracy and how algorithms should be structured in order to minimize the computing time, and introduces two basic tools on which most eigensolvers are based, namely similarity transformations and deflation. For accuracy reasons, orthogonal transformations should be used whenever possible. Some simple orthogonal transformations are discussed in Section 3.

Most eigenvalue solvers work in two phases: First the matrix is reduced to tridiagonal form and then the eigenvalue problem for the tridiagonal matrix is solved. Algorithms for these two phases are discussed in Sections 4 and 5, respectively, whereas Section 6 reviews algorithms that do not rely on an initial reduction. A synopsis of the available algorithms is given in Section 7. The methods presented here also carry over to complex Hermitean matrices. For simplicity we will focus on the real case. Finally, Section 8 points to relevant software.

1 Setting the Stage

After introducing some notational conventions, this section recalls the symmetric standard and generalized eigenvalue problems and points out the differences between direct and iterative eigenvalue solvers.

1.1 Some Notation

Throughout this article, matrices are denoted by uppercase letters $\mathsf{A}, \mathsf{B}, \ldots$, and the (i, j) entry of A is referred to as $\mathsf{A}(i, j)$. Analogously, lowercase letters $\mathbf{x}, \mathbf{y}, \ldots$ stand for (column) vectors with entries $\mathbf{x}(i)$, whereas greek letters $\alpha, \beta, \ldots$ denote scalars. Unless explicitly stated otherwise, all matrices are n-by-n and all vectors have length n.

I is the identity matrix (with entries $\mathsf{I}(i, j) = 1$ if $i = j$ and $\mathsf{I}(i, j) = 0$ otherwise), and $\mathsf{0}$ is the matrix with all entries equal to zero. For any matrix $\mathsf{A} \in \mathbb{R}^{m \times n}$, $\mathsf{A}^\top \in \mathbb{R}^{n \times m}$ denotes the *transpose* of A, i.e., $\mathsf{A}^\top(i, j) = \mathsf{A}(j, i)$ for all i, j. A square

matrix A is *symmetric* if $\mathsf{A}^\top = \mathsf{A}$.

1.2 The Symmetric Eigenvalue Problem

The *symmetric (standard) eigenvalue problem* consists of computing all or selected *eigenvalues* and associated *eigenvectors* of a symmetric matrix $\mathsf{A} \in \mathbb{R}^{n\times n}$, that is, scalars λ_i and vectors $\mathbf{q}_i \neq \mathbf{0}$ satisfying

$$\mathsf{A} \cdot \mathbf{q}_i = \mathbf{q}_i \cdot \lambda_i \ . \tag{1}$$

The eigenvalues of A are just the n roots (counting multiplicity) of its *characteristic polynomial* $p(\lambda) = \det(\mathsf{A} - \lambda\mathsf{I})$. (It is tempting to use this property for computing the eigenvalues: First determine the coefficients of the characteristic polynomial and then its roots. However, this method cannot be recommended because it gives highly inaccurate results.)

A pair $(\lambda_i, \mathbf{q}_i)$ satisfying Eq. (1) is called an *eigenpair* of A, and the set of all eigenvalues is called the *spectrum*, $\mathrm{spec}(\mathsf{A})$. For symmetric matrices all eigenvalues are real, and there exists a complete set of n mutually orthogonal, normalized (*orthonormal*, for brevity) real eigenvectors:

$$\mathbf{q}_i^\top \mathbf{q}_j = 0 \quad \text{for } i \neq j \qquad \text{and} \qquad \mathbf{q}_i^\top \mathbf{q}_i = 1 \quad \text{for } i = 1, \ldots, n \ .$$

Together with Eq. (1) this implies that A has an *eigendecomposition*

$$\mathsf{A} = \mathsf{Q}\mathsf{\Lambda}\mathsf{Q}^\top \ , \tag{2}$$

where $\mathsf{Q} = (\, \mathbf{q}_1 \,|\, \ldots \,|\, \mathbf{q}_n \,) \in \mathbb{R}^{n\times n}$ is an *orthogonal* matrix (i.e., $\mathsf{Q}^\top\mathsf{Q} = \mathsf{I}$), and

$$\mathsf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n) := \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \ .$$

By convention the eigenvalues are numbered ascendingly, that is, $\lambda_1 \leq \ldots \leq \lambda_n$.

Not all applications require computing the full eigendecomposition. Sometimes if suffices to compute only the eigenvalues, sometimes only selected eigenpairs are needed (e.g., eigenpairs for all non-negative eigenvalues or for the 100 largest eigenvalues), etc.

In the *symmetric generalized eigenvalue problem* one wants to compute eigenpairs $(\lambda_i, \mathbf{q}_i)$ satisfying

$$\mathsf{A} \cdot \mathbf{q}_i = \mathsf{B} \cdot \mathbf{q}_i \cdot \lambda_i \ ,$$

where B is another symmetric matrix. In the generalized case the eigenvectors need not be mutually orthogonal. In most applications B is also *positive definite* (i.e., $\mathbf{z}^\top\mathsf{B}\mathbf{z}\rangle 0$ for all $\mathbf{z} \neq \mathbf{0}$ or, equivalently, all eigenvalues of B are positive). In our treatment of the generalized problem we will focus on this case.

1.3 Direct and Iterative Eigensolvers

For the solution of linear systems we can choose between direct methods like Gaussian elimination, which give the solution after a fixed number of operations, and iterative solvers like the *conjugate gradients* method, which produce a sequence of increasingly accurate approximations to the solution. Iterative solvers terminate as soon as the required precision is attained and can thus lead to considerable savings in operations (and in memory as well).

By contrast, a result from Galois theory implies that *there can be no algorithm* that computes the eigenvalues of every matrix in a finite number of operations (additions, subtractions, multiplications, divisions, roots of any order). Thus, every eigensolver must have an iterative component.

Nevertheless some of the methods are called *direct solvers*. As with linear systems, direct eigensolvers *transform* the matrix to obtain the eigensystem, whereas (purely) *iterative solvers* work with the original matrix and try to extract selected eigenvalues and eigenvectors from appropriate low-dimensional subspaces of $\mathbb{R}^n$. Direct solvers are the methods of choice to compute a significant portion of the eigendecomposition for small to medium-sized matrices ($n \lesssim 5000$, say), while iterative solvers are used when only a few (up to 100, say) eigenpairs of very large — and typically sparse — matrices are sought. In this article we discuss only the direct solvers; iterative methods are treated in another contribution.

2 Eigenvalue Computations: How Good, How Fast, and How?

At the beginning of this section we introduce some more notations and definitions. Then we explain why eigenvalue solvers cannot compute the exact eigenvalues and eigenvectors of a matrix, and what kind of accuracy a "good" algorithm can achieve. Issues related to minimizing the computing time on today's high-performance computers are discussed subsequently. Finally, two basic tools for eigenvalue computations are introduced, namely similarity transformations that make the matrix in some way easier to handle, and deflation, which helps to break the problem into smaller ones whose solution is much cheaper.

2.1 More Notation

For contiguous portions of vectors and matrices we use the Matlab-style[30] colon notation: $\mathbf{x}(i_1 : i_2)$ denotes the length-$(i_2 - i_1 + 1)$ vector consisting of $\mathbf{x}$'s entries i_1 through i_2, and $\mathsf{A}(i_1 : i_2, j_1 : j_2)$ is the $(i_2 - i_1 + 1) \times (j_2 - j_1 + 1)$ matrix containing rows $i_1, \ldots, i_2$ of A's columns $j_1, \ldots, j_2$. An isolated colon stands for the whole index range in the respective direction. Thus, $\mathsf{A}(5, :)$ denotes the fifth row of A, whereas $\mathsf{A}(:, 3 : 7)$ contains columns 3 through 7 of the matrix.

The "size" of vectors and matrices is measured with *norms*. For our purposes, the *Euclidean norm*

$$\|\mathbf{x}\|_2 := \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_{i=1}^{n} \mathbf{x}(i)^2}$$

plays the dominant rôle. Its associated matrix norm is the *spectral norm*

$$\|\mathsf{A}\|_2 := \max\left\{\frac{\|\mathsf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \;:\; \mathbf{x} \neq \mathbf{0}\right\} \qquad (= \max\{|\lambda_1|, |\lambda_n|\} \text{ if } \mathsf{A} \text{ is symmetric}) \ .$$

Since computing the spectral norm of a matrix is very expensive, often the *Frobenius norm*

$$\|\mathsf{A}\|_F := \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \mathsf{A}(i,j)^2}$$

is used instead. Another important norm is the *maximum norm*

$$\|\mathbf{x}\|_\infty := \max\{|\,\mathbf{x}(i)| \;:\; i = 1, \ldots, n\} \ .$$

For iterative processes the *order of convergence* indicates how fast the desired values are approached. Let $\|\cdot\|$ denote a vector norm. A sequence of vectors $(\mathbf{x}_k)_{k\geq 0}$ converges *linearly* to a limit $\mathbf{x}^*$ if there is some constant $0\langle c\langle 1$ such that $\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq c \cdot \|\mathbf{x}_k - \mathbf{x}^*\|$ for all k. This means that a constant number of steps is required to obtain one additional correct digit in the approximation $\mathbf{x}_k$. The sequence converges *quadratically* (or *cubically*) if $\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq C \cdot \|\mathbf{x}_k - \mathbf{x}^*\|^2$ (or $\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq C \cdot \|\mathbf{x}_k - \mathbf{x}^*\|^3$) for some $C\rangle 0$. With quadratic and cubic convergence, the number of correct digits is *doubled* or *tripled* in each step. Thus $\mathbf{x}^*$ is approached very fast, once we have come somewhere close to it.

2.2 Accuracy Issues

Two obstacles prevent us from getting the *exact* eigenvalues and eigenvectors of a matrix on a computer.

First, in general the computation of the eigensystem involves an iterative process, which must be interrupted at some point before the correct values are reached, resulting in a so-called *truncation error*.

But even if this were not the case, the eigenvalues typically cannot be stored exactly in the finite number of memory cells that are allocated for each "real" number. The same holds for the results of most intermediate results. E.g., the IEEE standard 754[2] represents double-precision numbers with 64 bits, twelve of them encoding the sign and magnitude, and the remaining 52 bits holding the most significant binary digits of the number — corresponding to roughly 16 decimal places accuracy.

The best we may reasonably expect is that the computed result $\mathsf{fl}(\alpha \circ \beta)$ of each operation, where $\circ$ stands for addition, subtraction, multiplication, or division, is the floating-point number that is closest to the exact result $\alpha \circ \beta$, that is,

$$\mathsf{fl}(\alpha \circ \beta) = (\alpha \circ \beta) \cdot (1 + \epsilon) \ , \tag{3}$$

where ϵ is some tiny quantity depending on the operation $\circ$ and on the operands α and β. If the arithmetic of the computer conforms to the IEEE standard (this is true for all recent workstations and personal computers) then the quantities ϵ in Eq. (3) are guaranteed to be below some constant bound ε, the so-called *machine epsilon*. In double-precision, $\varepsilon \approx 2.22 \cdot 10^{-16}$.

But even if each single operation produces only a tiny relative error, a badly chosen (though mathematically correct) sequence of operations may yield an utterly wrong final result. In addition it is often very difficult to derive good estimates for the error of the computed quantity, the so-called *forward error*.

A much simpler technique for analyzing the behaviour of the algorithms was invented by Wilkinson[39]. Roughly speaking, his *backward error* analysis tries to put the blame for the rounding errors on the initial data. In the context of eigenvalue computations this means proving that the computed eigenvalues $\mathsf{fl}(\lambda_i)$ are the *exact* eigenvalues not of the original matrix A, but of a (slightly) perturbed matrix $\mathsf{A}+\Delta\mathsf{A}$. An eigenvalue algorithm is called *(backward) stable* if the perturbation $\Delta\mathsf{A}$ is very small compared to A:

$$\|\Delta\mathsf{A}\|_2 = \mathcal{O}(\varepsilon) \cdot \|\mathsf{A}\|_2 \ , \tag{4}$$

where $\mathcal{O}(\varepsilon)$ stands for a "small multiple" of the machine epsilon that may grow with a low-degree polynomial in the matrix dimension, like $4n\varepsilon$ or $2n^2\varepsilon$.

In order to derive bounds for the errors $|\mathsf{fl}(\lambda_i) - \lambda_i|$ (λ_i being the exact eigenvalues of A), the backward error analysis must be complemented by *perturbation analysis*, which investigates how much the eigenvalues can change if the matrix is perturbed. For the symmetric case, the following simple bound holds.

Theorem[18]. *Let* A *and* $\tilde{\mathsf{A}}$ *be symmetric matrices with eigenvalues* $\lambda_1 \le \ldots \le \lambda_n$ *and* $\tilde{\lambda}_1 \le \ldots \le \tilde{\lambda}_n$*, respectively. Then*

$$|\tilde{\lambda}_i - \lambda_i| \le \|\tilde{\mathsf{A}} - \mathsf{A}\|_2 \ . \tag{5}$$

The accumulated changes of all eigenvalues can be estimated with the Frobenius norm.

Theorem (Wielandt-Hoffman)[18]. *Let* A *and* $\tilde{\mathsf{A}}$ *be symmetric matrices with eigenvalues* $\lambda_1 \le \ldots \le \lambda_n$ *and* $\tilde{\lambda}_1 \le \ldots \le \tilde{\lambda}_n$*, respectively. Then*

$$\sqrt{\sum_{i=1}^{n} (\tilde{\lambda}_i - \lambda_i)^2} \le \|\tilde{\mathsf{A}} - \mathsf{A}\|_F \ .$$

The changes of eigenvectors corresponding to *simple* eigenvalues can be bounded as follows.

Theorem[11]. *Let* A *and* $\tilde{\mathsf{A}}$ *be symmetric matrices with eigenvalues* $\lambda_1 \le \ldots \le \lambda_n$ *and* $\tilde{\lambda}_1 \le \ldots \le \tilde{\lambda}_n$ *and associated orthonormal eigenvectors* $\mathbf{q}_1, \ldots, \mathbf{q}_n$ *and* $\tilde{\mathbf{q}}_1, \ldots, \tilde{\mathbf{q}}_n$*, respectively. Let* θ_i *denote the (acute) angle between* $\tilde{\mathbf{q}}_i$ *and* $\mathbf{q}_i$*. Then*

$$\frac{1}{2}\sin 2\theta_i \le \frac{\|\tilde{\mathsf{A}} - \mathsf{A}\|_2}{\min\{|\lambda_j - \lambda_i| \ : \ j \ne i\}} \qquad \text{if} \quad \lambda_j \ne \lambda_i \text{ for all } j \ne i \ . \tag{6}$$

Therefore the direction of an eigenvector $\mathbf{q}_i$ changes only little, provided that the matrix itself changes very little and that the associated eigenvalue λ_i is simple and well separated from the remaining eigenvalues. Note that when θ_i is small then $\frac{1}{2}\sin 2\theta_i \approx \sin\theta_i \approx \theta_i$.

Eqs. (4), (5), and (6) imply that stable algorithms can compute the eigenvalues of a symmetric matrix A with an error $\mathcal{O}(\varepsilon) \cdot \|\mathsf{A}\|_2$ and the eigenvectors corresponding to well-separated eigenvalues with an error $\mathcal{O}(\varepsilon)$. For the large eigenvalues

($|\lambda_i| \approx \|\mathsf{A}\|_2$) this means that the computed values will be correct except for a few digits at the end. For very small eigenvalues ($|\lambda_i| \approx \mathcal{O}(\varepsilon)\|\mathsf{A}\|_2$), however, very few correct figures can be guaranteed. Indeed, even the magnitude and the sign of the computed eigenvalue may be wrong.

In general, the bounds given in Eqs. (5) and (6) cannot be improved much. For particular classes of matrices, however, the dependence of the eigenvalues on the matrix elements is much stronger: Then *relative* perturbations of the matrix (i.e., each matrix entry is perturbed proportionally to its magnitude) lead to relative eigenvalue perturbations. In these situations appropriate algorithms can compute almost all digits even of extremely tiny eigenvalues[4].

2.3 Performance Issues

Among the stable methods we would like to select one that takes the least time to determine the desired eigenvalues and eigenvectors. The computing time depends on several factors, most notably on the number of operations, on the rate at which these operations can be performed, and on the potential for exploiting parallelism.

In the context of direct eigensolvers, the overall work is adequately captured by counting only the floating-point operations (flop: additions, subtractions, multiplications, divisions, and square roots) and ignoring everything else, like index manipulations etc.

The *performance* (or *execution rate*) of an algorithm is measured in Mflop/s (millions of flop per second). On today's machines with processors running at hundreds of MHz and main memories being almost an order of magnitude slower, the performance is mainly determined by the "data re-use factor" r, which is the number of operations performed, divided by the amount of data that are moved between the main memory and the processor (more precisely, between main memory and the caches, which are small, but fast memory buffers running almost at full processor speed). This issue is best explained with the different levels of the *BLAS* (*Basic Linear Algebra Subprograms*)[13,14,28].

The BLAS define a set of subroutines for performing some simple recurring tasks in linear algebra computations. To consider just three of these routines, the (double-precision) function `DDOT` returns the scalar product $\mathbf{x}^\top\mathbf{y}$ of two vectors. This routine does $2n-1$ flop and reads $2n$ elements from memory (the entries of the two vectors), yielding $r \approx 1$. Another routine, `DGEMV`, computes a matrix–vector product with roughly $2n^2$ flop and n^2 accesses to memory, thus $r \approx 2$. And finally, the routine for computing matrix–matrix products, `DGEMM`, requires about $4n^2$ memory accesses to do $2n^3$ flop, resulting in a much higher ratio $r \approx n/2$. (The "level 1 BLAS" do order-of-n flop on order-of-n data, the level 2 routines do order-of-n^2 flop on order-of-n^2 data, and the level 3 routines do order-of-n^3 flop on order-of-n^2 data.) As can be seen from Table 1, a higher re-use factor can significantly improve the performance. (Note that only the level 3 routine comes anywhere close to the processor's peak performance of 266Mflop/s.)

Therefore the algorithms should be (re)structured in such a way that a major part of their operations can be done with level 3 routines, even if the overall number of operations is slightly increased. Algorithms with this property are often called

Table 1. Performance of selected operations on a 266MHz PentiumII. All vectors had length $n = 1000$, all matrices were n-by-n.

Operation	BLAS Routine	re-use factor r	Mflop/s
$\alpha := \mathbf{x}^\top \mathbf{y}$	`DDOT` (BLAS 1)	1	36.1
$\mathbf{y} := \alpha \mathsf{A}\mathbf{x} + \beta\mathbf{y}$	`DGEMV` (BLAS 2)	2	61.1
$\mathsf{C} := \alpha\mathsf{A}\mathsf{B} + \beta\mathsf{C}$	`DGEMM` (BLAS 3)	$n/2$	195.2

blocked algorithms.

Modern high-performance computers typically feature multiple processors. To fully utilize the potential power of these machines, *parallelism* must be exploited, i.e., the work (and in most cases the data, too) must be distributed among the processors. In particular, the algorithm must contain enough *independent* operations to keep all the processors busy. In addition, a processor cannot work completely on its on, but from time to time it must *synchronize* with other processors in order to exchange information. As synchronization causes significant administrative overhead, one should strive for so-called *coarse-grained parallelism*, where synchronizations occur rarely, thus allowing each processor to do a lot of "useful" operations between them — as opposed to *fine-grained parallelism* with frequent synchronizations and just a few flop in between.

Algorithms that are based mainly on level 3 operations lend themselves in a natural way to coarse-grained parallelism. This is another argument in favor of blocking, besides the fact that these algorithms also achieve high per-node computing performance.

2.4 Similarity Transformations

Almost every direct method for computing eigensystems makes use of the fact that eigenvalues are invariant under *similarity transformations* $\mathsf{A} \mapsto \mathsf{X}^{-1}\mathsf{A}\mathsf{X} =: \tilde{\mathsf{A}}$, where X denotes an arbitrary non-singular matrix. More precisely, if $(\lambda, \mathbf{x})$ is an eigenpair of A then $\mathsf{A}\mathbf{x} = \mathbf{x}\lambda$, and hence

$$(\mathsf{X}^{-1}\mathsf{A}\mathsf{X}) \cdot (\mathsf{X}^{-1}\mathbf{x}) = (\mathsf{X}^{-1}\mathbf{x}) \cdot \lambda \ .$$

Thus $(\lambda, \mathsf{X}^{-1}\mathbf{x})$ is an eigenpair of the transformed matrix $\tilde{\mathsf{A}}$.

Most eigensolvers exploit this property in the following way: First, a suitable similarity transformation $\mathsf{A} \mapsto \mathsf{X}^{-1}\mathsf{A}\mathsf{X} =: \tilde{\mathsf{A}}$ reduces A to a matrix $\tilde{\mathsf{A}}$ that is in some way more easily handled (see Sections 4 and 5), then the desired eigenvalues $\tilde{\lambda}_i$ and associated eigenvectors $\tilde{\mathbf{x}}_i$ of $\tilde{\mathsf{A}}$ are computed, and finally these are back-transformed into the eigenvalues $\lambda_i = \tilde{\lambda}_i$ and eigenvectors $\mathbf{x}_i = \mathsf{X} \cdot \tilde{\mathbf{x}}_i$ of the original matrix A.

Whenever possible, *orthogonal* similarity transformations $\mathsf{A} \mapsto \mathsf{Q}^\top\mathsf{A}\mathsf{Q}$ (Q being an orthogonal matrix) should be used, for three reasons. First, the inverse $\mathsf{Q}^{-1} = \mathsf{Q}^\top$ is readily available. Second, these transformations preserve symmetry: If A is symmetric then $\mathsf{Q}^\top\mathsf{A}\mathsf{Q}$ is symmetric, too. And third, orthogonal transformations are very *stable* in the sense that they induce a small backward error. It is

a nice feature of the symmetric (standard) eigenvalue problem that in principle all computations can be done with orthogonal transformations.

2.5 Deflation

Suppose that we have found an orthogonal matrix Q such that

$$\mathsf{Q}^\top \mathsf{A}\mathsf{Q} = \begin{pmatrix} \mathsf{A}_1 & 0 \\ 0 & \mathsf{A}_2 \end{pmatrix} \tag{7}$$

with $\mathsf{A}_1 \in \mathbb{R}^{k\times k}$ and $\mathsf{A}_2 \in \mathbb{R}^{(n-k)\times(n-k)}$. Then we have $\operatorname{spec}(\mathsf{A}) = \operatorname{spec}(\mathsf{A}_1) \cup \operatorname{spec}(\mathsf{A}_2)$, and therefore the eigenvalues of A can be obtained by computing those of the smaller matrices A_i. This reduction of the size of the matrices that must be transformed further — the so-called *deflation* — leads to considerable flop savings.

The eigenvectors of A may also be obtained from those of the smaller matrices A_i. To this end we partition Q accordingly: $\mathsf{Q} = (\,\mathsf{Q}_1 \,|\, \mathsf{Q}_2\,)$, where Q_1 consists of the first k columns in Q. Then Eq. (7) is equivalent to $\mathsf{A}\mathsf{Q}_i = \mathsf{Q}_i\mathsf{A}_i$, $i = 1, 2$. Given an eigenpair $(\tilde\lambda, \tilde{\mathbf{x}})$ of one of the smaller matrices A_i, we thus obtain

$$\mathsf{A}\cdot(\mathsf{Q}_i\tilde{\mathbf{x}}) = \mathsf{Q}_i\cdot(\mathsf{A}_i\tilde{\mathbf{x}}) = (\mathsf{Q}_i\tilde{\mathbf{x}})\cdot\lambda\ ,$$

which means that $(\lambda, \mathsf{Q}_i\tilde{\mathbf{x}})$ is an eigenpair of A.

3 Tools of the Trade: Basic Orthogonal Transformations

Orthogonal matrices play an important rôle in the computation of eigenvalues and eigenvectors (e.g., the eigenvector matrix Q in the eigendecomposition (2) is orthogonal). These orthogonal matrices are built up from two basic types, rotations and Householder transformations, which are introduced in this section. We also discuss techniques for improving the performance by applying these basic transformations in a blocked fashion.

3.1 Rotations

An (i, j) *rotation* in $\mathbb{R}^n$ is given by the matrix

$$\mathsf{R} = \mathsf{R}(i,j,\theta) = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & & s & & \leftarrow i \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & -s & & & c & & \leftarrow j \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & & & & & & & & & 1 \end{pmatrix} \in \mathbb{R}^{n\times n}, \tag{8}$$

where $c = \cos\theta$, $s = \sin\theta$, and all the remaining entries of R are zero. It is easy to verify that R is orthogonal.

Applying R to some column vector $\mathbf{x} \in \mathbb{R}^n$ via $\mathbf{x} \mapsto \mathsf{R}^\top \cdot \mathbf{x}$ corresponds to a counter-clockwise rotation by the angle θ in the (i, j) coordinates plane. The same is achieved for row vectors $\mathbf{x}^\top$ via $\mathbf{x}^\top \mapsto \mathbf{x}^\top \cdot \mathsf{R}$. In both cases only the ith and jth entries of $\mathbf{x}$ change.

By letting

$$c = \frac{\mathbf{x}(i)}{\sqrt{\mathbf{x}(i)^2 + \mathbf{x}(j)^2}} \qquad \text{and} \qquad s = \frac{-\mathbf{x}(j)}{\sqrt{\mathbf{x}(i)^2 + \mathbf{x}(j)^2}} \tag{9}$$

in the above transformations we can zero the jth entry of $\mathbf{x}$. (Note that only the cosine and the sine of the rotation angle θ are present in the formulas (8) and (9); there is no need to compute the angle itself.)

Applying the rotation to a matrix $\mathsf{A} \in \mathbb{R}^{n\times m}$ from the left, $\mathsf{A} \mapsto \mathsf{R}^\top \cdot \mathsf{A}$, affects only the ith and jth row of A,

$$\begin{aligned} \mathsf{tmp} &:= c \cdot \mathsf{A}(i,:) + s \cdot \mathsf{A}(j,:) \\ \mathsf{A}(j,:) &:= -s \cdot \mathsf{A}(i,:) + c \cdot \mathsf{A}(j,:) \\ \mathsf{A}(i,:) &:= \mathsf{tmp} \end{aligned}$$

whereas $\mathbb{R}^{m\times n} \ni \mathsf{A} \mapsto \mathsf{A} \cdot \mathsf{R}$ does the same with columns i and j. In either case, the transformation takes $4m$ multiplications and $2m$ additions. If a symmetric matrix A is transformed from both sides, $\mathsf{A} \mapsto \mathsf{R}^\top \cdot \mathsf{A} \cdot \mathsf{R}$, then the resulting matrix is again symmetric. Therefore the transformations must be applied only to the lower (or upper) triangle of A, thereby approximately halving the overall cost to $6n$ flop.

Rotations are very stable. It can be shown[40] that the backward error corresponding to a transformation $\mathsf{A} \mapsto \mathsf{R}^\top \cdot \mathsf{A}$ is bounded by $6\varepsilon\|\mathsf{A}\|_F$.

There are variants of rotations — called *fast Givens rotations*[35] — that require only half as many multiplications as the "ordinary" rotations described above, resulting in $4m$ flop for the transformation $\mathsf{A} \mapsto \mathsf{R}^\top\mathsf{A}$. These savings are achieved by an appropriate (implicit) scaling of the matrix R.

In rotation–based algorithms, parallelism can be exploited in two ways. Either the work of each rotation is split among the processors (yielding a rather fine-grained parallelism), or in some situations several rotations in disjoint planes can be applied simultaneously[24].

3.2 Householder Transformations

While rotations may be used to introduce single zeros in a vector or a matrix, Householder transformations can zero out more than one entry.

A length-n *Householder transformation* is described by the matrix

$$\mathsf{H} = \mathsf{H}(\mathbf{y}) = \mathsf{I} - \mathbf{y}\tau\mathbf{y}^\top \in \mathbb{R}^{n\times n}, \tag{10}$$

where $\mathbf{y} \in \mathbb{R}^n$ is an arbitrary vector and

$$\tau = \begin{cases} 0, & \text{if } \mathbf{y} = 0 \\ 2/\|\mathbf{y}\|_2^2, & \text{otherwise} \end{cases} \in \mathbb{R}.$$

A short computation reveals that H is orthogonal and that the transformations $\mathbf{x} \mapsto \mathsf{H}^\top \cdot \mathbf{x}$ and $\mathbf{x}^\top \mapsto \mathbf{x}^\top \cdot \mathsf{H}$ correspond to a reflection of $\mathbf{x}$ at the (hyper)plane perpendicular to $\mathbf{y}$, thus inverting $\mathbf{x}$'s component in direction $\mathbf{y}$. (Note that H is symmetric; thus we might write H instead of $\mathsf{H}^\top$.)

To zero out the entries $i+1, \ldots, j$ of a given vector $\mathbf{x} \in \mathbb{R}^n$, we choose

$$\mathbf{y} = (0, \ldots, 0, \mathbf{x}(i) \pm \|\mathbf{x}(i+1:j)\|_2, \mathbf{x}(i+1), \ldots, \mathbf{x}(j), 0, \ldots, 0)^\top \ , \tag{11}$$

where "$\pm$" is chosen equal to the sign of $\mathbf{x}(i)$ in order to avoid loss of precision through cancellation. This choice gives

$$\mathsf{H}^\top \cdot \mathbf{x} = (\mathbf{x}(1), \ldots, \mathbf{x}(i-1), \mp\|\mathbf{x}(i:j)\|_2, 0, \ldots, 0, \mathbf{x}(j+1), \ldots, \mathbf{x}(n))^\top \ .$$

In particular, only the entries $i, \ldots, j$ of $\mathbf{x}$ are affected by the transformation.

Analogously the transformations $\mathbb{R}^{n\times m} \ni \mathsf{A} \mapsto \mathsf{H}^\top \cdot \mathsf{A}$ and $\mathbb{R}^{m\times n} \ni \mathsf{A} \mapsto \mathsf{A} \cdot \mathsf{H}$ affect only rows (columns, resp.) $i, \ldots, j$ of A: Making use of Eqs. (10) and (11), in the first case one computes

$$\mathbf{z}^\top := \tau \cdot \mathbf{y}(i:j)^\top \cdot \mathsf{A}(i:j,:) \in \mathbb{R}^{1\times m} \tag{12}$$

$$\mathsf{A}(i:j,:) := \mathsf{A}(i:j,:) - \mathbf{y}(i:j) \cdot \mathbf{z}^\top. \tag{13}$$

The matrix–vector product (12) and the rank-1-update (13) each require approximately $2\ell m$ operations ($\ell = j - i + 1$ is the "active length" of the transformation), for a total of $4\ell m$ flop. (Note that one *never* computes $\mathsf{H}^\top \cdot \mathsf{A}$ as a matrix–matrix product, which would require approximately $2\ell^2 m$ flop.)

As with rotations, symmetry can be exploited to save operations. The two-sided transformation $\mathbb{R}^{n\times n} \ni \mathsf{A} \mapsto \mathsf{H}^\top \cdot \mathsf{A} \cdot \mathsf{H}$ of a symmetric matrix A is performed as follows:

$$\mathbf{z} := \mathsf{A} \cdot \mathbf{y} \cdot \tau \tag{14}$$

$$\mathbf{v} := \mathbf{z} - \mathbf{y} \cdot \frac{\tau(\mathbf{y}^\top \mathbf{z})}{2}$$

$$\mathbf{A} := \mathbf{A} - \mathbf{y} \cdot \mathbf{v}^\top - \mathbf{v} \cdot \mathbf{y}^\top \ . \tag{15}$$

Due to the symmetry, only one triangle of A must be updated in the symmetric rank-2-update (15), provided that only one triangle is used in the symmetric matrix–vector product (14). Again, the two-sided transformation of A can be effected at the same cost as a one-sided transformation of a non-symmetric matrix, that is, $4n^2$ flop.

Householder transformations are very stable, too[20]. The backward error corresponding to a transformation $\mathsf{A} \mapsto \mathsf{H}^\top \cdot \mathsf{A}$ is bounded by $cn\varepsilon\|\mathsf{A}\|_F$, where c is some small constant.

With Householder transformations, parallelism is exploited by working on a distributed matrix A and doing each of the steps (14) and (15) in parallel.

3.3 Blocked Householder Transformations

Applying single rotations or Householder transformations does not allow using any level 3 BLAS. The situation changes if a *sequence* of transformations must be applied to the same matrix.

Bischof and Van Loan[6] discovered that the product $\mathsf{Q} = \mathsf{H}_1 \cdot \ldots \cdot \mathsf{H}_k$ of k length-n Householder transformations $\mathsf{H}_j = \mathsf{I} - \mathbf{y}_j \tau_j \mathbf{y}_j^\top$ can be written in the form

$$\mathsf{Q} = \mathsf{I} - \mathsf{W}\mathsf{Y}^\top \qquad (\textit{WY representation}), \tag{16}$$

where W and Y are suitable n-by-k matrices.

For $k = 1$ the representation (16) is obviously valid with $\mathsf{W} = \mathbf{y}_1 \tau_1$ and $\mathsf{Y} = \mathbf{y}_1$. To proceed from k to $k+1$, we assume that the matrices W and Y in (16) are known and that $\tilde{\mathsf{Q}} = \mathsf{H}_1 \cdot \ldots \cdot \mathsf{H}_{k+1} = \mathsf{Q} \cdot \mathsf{H}_{k+1}$. Then a short calculation reveals that $\tilde{\mathsf{Q}} = \mathsf{I} - \tilde{\mathsf{W}}\tilde{\mathsf{Y}}^\top$ with the n-by-$(k+1)$ matrices

$$\tilde{\mathsf{W}} = (\,\mathsf{W} \,|\, \mathsf{Q}\mathbf{y}_{k+1}\tau_{k+1}\,) \qquad \text{and} \qquad \tilde{\mathsf{Y}} = (\,\mathsf{Y} \,|\, \mathbf{y}_{k+1}\,)\ .$$

Thus each additional transformation H_j requires only appending one new column to W and Y. Note that in particular the columns of Y are just the vectors $\mathbf{y}_j$ defining the transformations H_j.

Given the representation (16), applying the k transformations to a matrix A amounts to two matrix–matrix products,

$$\mathsf{H}_k^\top \cdot \ldots \cdot \mathsf{H}_1^\top \cdot \mathsf{A} = \mathsf{Q}^\top \cdot \mathsf{A} = \mathsf{A} - \mathsf{Y} \cdot (\mathsf{W}^\top \cdot \mathsf{A})\ ,$$

which performs significantly better than applying the single Householder transformations and requires only marginally more operations.

Later, Schreiber and Van Loan[36] refined the WY representation to

$$\mathsf{Q} = \mathsf{I} - \mathsf{Y}\mathsf{T}\mathsf{Y}^\top \qquad (\textit{compact WY representation}), \tag{17}$$

where Y is again n-by-k and T is an upper triangular k-by-k matrix. Here, the case $k = 1$ is covered by setting $\mathsf{Y} = \mathbf{y}_1$ and $\mathsf{T} = \tau_1$, and the step from k to $k+1$ is done by letting

$$\tilde{\mathsf{Y}} = (\,\mathsf{Y} \,|\, \mathbf{y}_{k+1}\,) \qquad \text{and} \qquad \tilde{\mathsf{T}} = \left(\begin{array}{c|c} \mathsf{T} & -\tau_{k+1}\mathsf{T}\mathbf{y}_{k+1} \\ \hline \mathbf{0}^\top & \tau_{k+1} \end{array}\right)\ .$$

The compact WY representation costs significantly less additional storage than the original WY representation does (T instead of W), but applying the transformations in the compact representation requires a third matrix–matrix product.

Under favorable conditions, applying a sequence of rotations to a matrix can also be organized in such a way that most of the work is done in matrix–matrix products[27].

4 Phase I: Reduction to Tridiagonal Form

Most eigenvalue solvers first reduce the symmetric matrix A to a symmetric tridiagonal matrix T (i.e., $\mathsf{T}(i,j) = 0$ whenever $|i-j|\rangle 1$) in order to make the ensuing iterative process simpler and cheaper. This section first presents the standard algorithm for tridiagonalizing a full matrix, as well as a blocked variant of the algorithm. Then the back-transformation of the eigenvectors is discussed, which is a final step after computing the eigenvalues and eigenvectors of the tridiagonal matrix T (cf. Section 5). Specialized reduction methods for banded matrices and generalized eigenvalue problems are described at the end of the section.

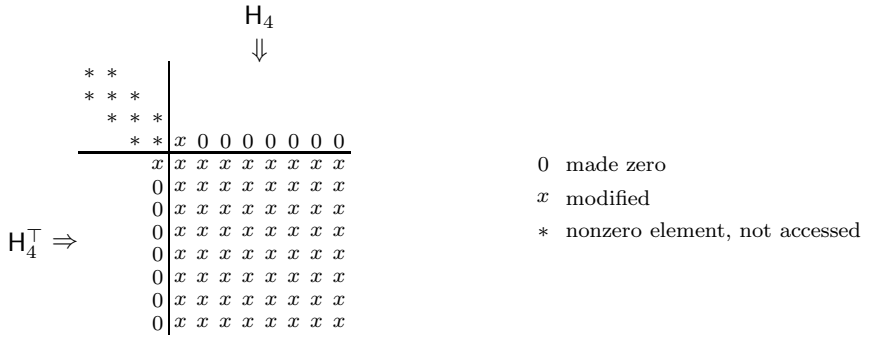

Figure 1. The fourth step in the reduction of a 12-by-12 matrix to tridiagonal form.

4.1 Householder Tridiagonalization

The standard algorithm for reducing an n-by-n symmetric full matrix to tridiagonal form, $\mathsf{A} \mapsto \mathsf{Q}_1^\top \mathsf{A}\mathsf{Q}_1 = \mathsf{T}$, proceeds in $n-2$ steps:

$$\begin{aligned} \mathsf{A} =: \mathsf{A}_0 \;\mapsto\; \mathsf{H}_1^\top \mathsf{A}_0 \mathsf{H}_1 =: \mathsf{A}_1 \;\mapsto\; \mathsf{H}_2^\top \mathsf{A}_1 \mathsf{H}_2 =: \mathsf{A}_2 \;\mapsto \\ \mapsto\; \ldots \;\mapsto\; \mathsf{H}_{n-2}^\top \mathsf{A}_{n-3} \mathsf{H}_{n-2} =: \mathsf{A}_{n-2} = \mathsf{T}\ , \end{aligned} \tag{18}$$

the kth step transforming column and row k to the desired shape with a suitable Householder transformation H_k, cf. Figure 1 and Algorithm 1.

Algorithm 1 Householder reduction to tridiagonal form.

for $k = 1$ **to** $n-2$
  determine a Householder transformation $\mathsf{H}_k = \mathsf{I} - \mathbf{y}_k \tau_k \mathbf{y}_k^\top$ that reduces
    $\mathsf{A}(k+1:n, k)$ to the form $(x, 0, \ldots, 0)^\top$
  $\mathbf{z}_k := \mathsf{A}\mathbf{y}_k \tau_k$
  $\mathbf{v}_k := \mathbf{z}_k - \mathbf{y}_k \cdot (\tau_k (\mathbf{y}_k^\top \mathbf{z}_k)/2)$
  $\mathsf{A} := \mathsf{A} - \mathbf{y}_k \cdot \mathbf{v}_k^\top - \mathbf{v}_k \cdot \mathbf{y}_k^\top$

Note that only the submatrix $\mathsf{A}(k+1:n, k+1:n)$ (and, due to symmetry, only one triangle of this matrix) is modified in the last line of the algorithm. Thus the reduction requires approximately $\frac{4}{3}n^3$ flop, which are done mainly within the level 2 BLAS.

Before Householder developed the method just described, the tridiagonalization had been done with a rotation–based algorithm due to Givens[40]. Householder's approach is superior for full matrices because it requires fewer operations, whereas Givens' algorithm can make better use of sparsity.

4.2 Blocked Householder Tridiagonalization

It is not necessary to build all the intermediate matrices A_k from Eq. (18) explicitly[15]. Instead, they can be represented in the so-called *factored form*

$$\mathsf{A}_s = \mathsf{A}_0 - \mathbf{y}_1\mathbf{v}_1^\top - \mathbf{v}_1\mathbf{y}_1^\top - \ldots - \mathbf{y}_s\mathbf{v}_s^\top - \mathbf{v}_s\mathbf{y}_s^\top = \mathsf{A}_0 - \mathsf{Y}_s\mathsf{V}_s^\top - \mathsf{V}_s\mathsf{Y}_s^\top \ , \qquad (19)$$

where the vectors $\mathbf{y}_k$ and $\mathbf{v}_k$ are defined as in Algorithm 1, and $\mathsf{Y}_s = (\,\mathbf{y}_1 \,|\, \ldots \,|\, \mathbf{y}_s\,)$ and $\mathsf{V}_s = (\,\mathbf{v}_1 \,|\, \ldots \,|\, \mathbf{v}_s\,)$ are n-by-s matrices. If the matrices A_k are built only every n_bth step then we arrive at Algorithm 2 (lines 4 through 6 and line 8 of the algorithm reflect the fact that $\mathbf{v}_k$ and $\mathbf{z}_k$ are computed from A_{k-1} and not from A_{s-1}, which is currently held in A).

Algorithm 2 Blocked Householder tridiagonalization.

for $s = 1$ **to** $n-2$ **step** n_b
  $\mathsf{Y} := ()$, $\mathsf{V} := ()$ { matrices with 0 columns }
  for $k = s$ **to** $\min\{s+n_b-1, n-2\}$
    if $k \rangle s$
      compute the kth column of A_{k-1} according to Eq. (19):
        $\mathsf{A}(:,k) := \mathsf{A}(:,k) - \mathsf{Y}\mathsf{V}^\top(:,k) - \mathsf{V}\mathsf{Y}^\top(:,k)$
    determine the Householder transformation H_k as in Algorithm 1
    $\mathbf{z}_k := \mathsf{A}\mathbf{y}_k\tau_k - \mathsf{Y}(\mathsf{V}^\top\mathbf{y}_k\tau_k) - \mathsf{V}(\mathsf{Y}^\top\mathbf{y}_k\tau_k)$
    compute $\mathbf{v}_k$ as in Algorithm 1
    $\mathsf{Y} := (\,\mathsf{Y}\,|\,\mathbf{y}_k\,)$, $\mathsf{V} := (\,\mathsf{V}\,|\,\mathbf{v}_k\,)$
  $\mathsf{A} := \mathsf{A} - \mathsf{Y}\mathsf{V}^\top - \mathsf{V}\mathsf{Y}^\top$ { after n_b steps rebuild A according to Eq. (19) }

Note that the rank-2-updates (last line in Algorithm 1) have been replaced with matrix–matrix products, whereas the symmetric matrix–vector products $\mathsf{A}\cdot\mathbf{y}$ in the computation of $\mathbf{z}_k$ persist. Therefore, the blocked algorithm does roughly one half of its $\frac{4}{3}n^3$ operations with level-3 BLAS while the remaining operations are still confined to the level-2 BLAS.

The portion of matrix–matrix operations can be further increased if the reduction to tridiagonal form is done in two phases[5]. First the matrix is reduced to banded form (almost completely with level 3 BLAS), and then the banded matrix is tridiagonalized (no level 3 BLAS, but significantly lower flop count than for the first phase). This approach typically outperforms the direct tridiagonalization if *no eigenvectors* are required.

4.3 Back-Transformation of the Eigenvectors

After the reduction of A to tridiagonal form, $\mathsf{A} \mapsto \mathsf{Q}_1^\top\mathsf{A}\mathsf{Q}_1 = \mathsf{T}$, all or selected eigenpairs $(\lambda_i, \mathbf{v}_i)$ of the tridiagonal matrix T are computed. While the eigenvalues λ_i of T are also eigenvalues of A, the associated eigenvectors must be back-transformed in order to obtain A's eigenvectors via $\mathbf{v}_i \mapsto \mathsf{Q}_1\cdot\mathbf{v}_i = \mathbf{q}_i$, cf. Section 2.4.

Let $\mathsf{H}_k = \mathsf{I} - \mathbf{y}_k\tau_k\mathbf{y}_k^\top$, $k = 1, \ldots, n-2$, be the Householder transformations that were used in the reduction $\mathsf{A} \mapsto \mathsf{T}$, i.e., $\mathsf{Q}_1 = \mathsf{H}_1 \cdot \ldots \cdot \mathsf{H}_{n-2}$, and let $\mathsf{V} =$

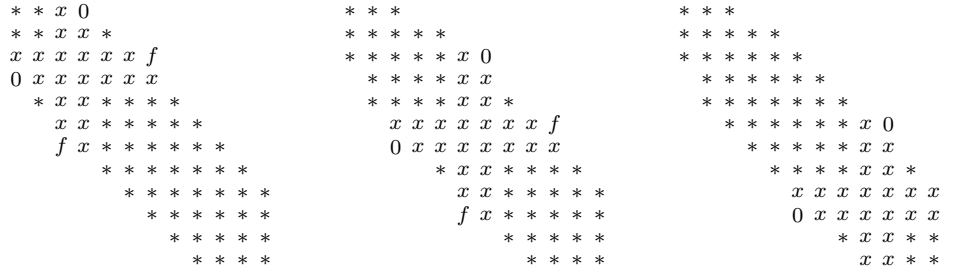

Figure 2. Rotations for zeroing the $(4,1)$ entry and for chasing the intermediate fill-in elements (denoted by f) in Schwarz' algorithm.

$(\,\mathbf{v}_1 \,|\, \ldots \,|\, \mathbf{v}_m\,)$ denote those eigenvectors of T that were computed and must be back-transformed. Then the corresponding eigenvectors of A are obtained as

$$(\,\mathbf{q}_1 \,|\, \ldots \,|\, \mathbf{q}_m\,) =: \mathsf{Q} = \mathsf{Q}_1 \cdot \mathsf{V} = \mathsf{H}_1 \cdot \ldots \cdot \mathsf{H}_{n-2} \cdot \mathsf{V}\ .$$

In contrast to the tridiagonal reduction, almost all of the roughly $2n^2m$ operations can be done with matrix–matrix products if we resort to the (compact or original) WY representation for applying $n_b \rangle 1$ transformations at a time, cf. Algorithm 3.

Algorithm 3 Blocked back-transformation of selected eigenvectors.

$\mathsf{Q} := \mathsf{V}$
for $s = n-2$ **to** 1 **step** $-n_b$
  determine the compact WY representation for the next $n_b' = \min\{n_b, s\}$
    transformations: $\mathsf{H}_{s-n_b'+1} \cdot \ldots \cdot \mathsf{H}_s = \mathsf{I} - \mathsf{YTY}^\top$
  apply these transformations via $\mathsf{Q} := \mathsf{Q} - \mathsf{YTY}^\top\mathsf{Q}$

Like any algorithm involving mainly products of large matrices, the back-transformation can be easily and efficiently parallelized.

4.4 Reduction of Banded Matrices

A symmetric matrix A is *banded* with *semibandwidth* b if $\mathsf{A}(i,j) = 0$ whenever $|i-j| \rangle b$. For narrow-banded matrices ($b \ll n$), Algorithms 1 and 2 are not optimal because they completely destroy the sparsity.

Such matrices are reduced with a rotation–based algorithm by Schwarz[37]. To understand how this algorithm works we consider a 12-by-12 matrix with semibandwidth $b = 3$, see Figure 2.

First, the outmost entry $(4,1)$ in the first column is made zero with a rotation $\mathsf{A} \mapsto \mathsf{R}^\top\mathsf{AR}$ in the $(3,4)$ plane (left picture in Figure 2). This creates a new *fill-in* entry f at position $(7,3)$, just outside the band. Then a second rotation in the $(6,7)$ plane is used to remove the fill-in entry (center picture), only to have another fill element appear at position $(10,6)$, which in turn is removed by the next rotation,

and so on. Each rotation "chases" the fill element b positions down the band until the end of the matrix is reached and no further fill-in is created (right picture).

Then we can zero the next entry $(3,1)$ in the first column with a $(2,3)$ rotation, again followed by a sequence of rotations for chasing the fill-in. When the first column of A is reduced to tridiagonal form we repeat the procedure for the second column, and so on. The whole method is summarized in Algorithm 4.

Algorithm 4 Schwarz' algorithm for tridiagonalizing banded matrices.

for $j = 1$ **to** $n-2$ { proceed by columns }
  for $d = \min\{b, n-j\}$ **to** 2 **step** -1 { zero the entry in the dth subdiagonal }
  make $\mathsf{A}(j+d, j)$ zero with a suitable rotation in the $(j+d-1, j+d)$ plane
  while the most recent rotation created a fill-in entry at some position (k, ℓ)
    make this entry zero by a suitable rotation in the $(k-1, k)$ plane

If A's eigenvectors are needed, too, then in the banded case the orthogonal matrix Q_1 (cf. Section 4.3), which is the product of all rotations that are used for the reduction, is built explicitly *during* the reduction. This is achieved by applying each plane-$(k-1,k)$ rotation R not only to the banded matrix, $\mathsf{A} \mapsto \mathsf{R}^\top \mathsf{A} \mathsf{R}$, but also to the columns $k-1$ and k of an n-by-n matrix Q_1 via $\mathsf{Q}_1 \mapsto \mathsf{Q}_1 \mathsf{R}$, where Q_1 has been initialized as the identity matrix.

The reduction of A requires $6bn^2$ flop. If the eigenvectors of A must be computed then the costs for accumulating Q_1, $3n^3$ flop, by far dominate the reduction costs. In this method some parallelism can be exploited by applying several rotations simultaneously[24].

A more recent reduction algorithm is based on Householder transformations[32], each transformation affecting just b rows and columns of A. This method allows coarser-grained parallelism than Schwarz' algorithm[26], and the accumulation of Q_1 can be done in a blocked fashion[5].

4.5 Reduction of the Generalized Eigenvalue Problem

The symmetric generalized eigenvalue problem $\mathsf{A}\mathbf{q}_i = \mathsf{B}\mathbf{q}_i\lambda_i$ with a symmetric *positive definite* matrix B can be transformed into a symmetric standard eigenvalue problem as follows. Let

$$\mathsf{B} = \mathsf{L}\mathsf{L}^\top \ , \tag{20}$$

where L is a lower triangular matrix with positive diagonal entries, be the *Cholesky decomposition* of B. (Algorithms for computing the *Cholesky factor* L are given below.) Then the condition $\mathsf{A}\mathbf{q}_i = \mathsf{B}\mathbf{q}_i\lambda_i$ is equivalent to

$$(\mathsf{L}^{-1}\mathsf{A}\mathsf{L}^{-\top}) \cdot (\mathsf{L}^\top \mathbf{q}_i) = (\mathsf{L}^{-1}\mathsf{B}\mathsf{L}^{-\top}) \cdot (\mathsf{L}^\top \mathbf{q}_i) \cdot \lambda_i = (\mathsf{L}^\top \mathbf{q}_i) \cdot \lambda_i \ ,$$

where $\mathsf{L}^{-\top}$ is a shorthand for $(\mathsf{L}^\top)^{-1}$ $(= (\mathsf{L}^{-1})^\top)$. Therefore, an eigenpair $(\lambda_i, \mathbf{q}_i)$ of the generalized eigenvalue problem corresponds to the eigenpair $(\lambda_i, \mathsf{L}^\top \mathbf{q}_i)$ of the symmetric standard eigenvalue problem for the matrix $\mathsf{M} = \mathsf{L}^{-1}\mathsf{A}\mathsf{L}^{-\top}$. Thus we arrive at Algorithm 5, which takes approximately $12n^3$ flop for solving the

generalized eigenvalue problem. (Note that the explicit calculation of L^{-1} and $\mathsf{L}^{-\top}$ is avoided by solving triangular systems with multiple right-hand sides, which is implemented in the level 3 BLAS routine `DTRSM`.)

Algorithm 5 Generalized eigenvalue problem via Cholesky decomposition of B.

compute the Cholesky decomposition $\mathsf{B} = \mathsf{L}\mathsf{L}^\top$ { Algorithm 7 }
compute $\mathsf{M} = \mathsf{L}^{-1}\mathsf{A}\mathsf{L}^{-\top}$ by solving two triangular systems with multiple right-hand sides, $\mathsf{X}\mathsf{L}^\top = \mathsf{A}$ (for X), and $\mathsf{L}\mathsf{M} = \mathsf{X}$ (for M)
use blocked Householder tridiagonalization (Algorithm 2) and an algorithm from Section 5 to compute the eigendecomposition $\mathsf{M} = \tilde{\mathsf{Q}}\Lambda\tilde{\mathsf{Q}}^\top$
compute the eigenvectors $\mathbf{q}_i = \mathsf{L}^{-\top}\tilde{\mathbf{q}}_i$ of the generalized problem by solving the triangular system $\mathsf{L}^\top\mathsf{Q} = \tilde{\mathsf{Q}}$ for Q

In general, the matrix M will be full even if A and B (and therefore B's Cholesky factor L, too) are *banded*. An algorithm by Crawford[9] avoids building M explicitly by interleaving its computation with the ensuing reduction to tridiagonal form. This leads to considerable flop and memory savings.

If B is ill-conditioned (i.e., its eigenvalues vary over many orders of magnitude) then severe loss of accuracy may happen because the backward error of Algorithm 5 grows with $\|\mathsf{B}^{-1}\|_2$. In this case the Cholesky decomposition should be replaced with the *eigendecomposition* $\mathsf{B} = \mathsf{S}\Delta^2\mathsf{S}^\top = (\mathsf{S}\Delta)(\mathsf{S}\Delta)^\top$, where S is orthogonal and Δ contains the roots of B's (positive!) eigenvalues. This leads to $\mathsf{M} = \Delta^{-1}\mathsf{S}^\top\mathsf{A}\mathsf{S}\Delta^{-1}$. In practice, the use of orthogonal matrices gives better results, albeit at higher cost.

The generalized eigenvalue problem with a matrix B that is not positive definite requires completely different techniques[18,34].

Let us finally give two algorithms for computing the Cholesky decomposition. Eq. (20) is equivalent to

$$\begin{aligned}\mathsf{B}(1,1) &= \mathsf{L}(1,1)\cdot\mathsf{L}(1,1)\ ,\\ \mathsf{B}(2:n,1) &= \mathsf{L}(2:n,1)\cdot\mathsf{L}(1,1)\ ,\text{ and}\\ \mathsf{B}(2:n,2:n) &= \mathsf{L}(2:n,2:n) + \mathsf{L}(2:n,1)\cdot\mathsf{L}(2:n,1)^\top\ ,\end{aligned}$$

i.e., $\mathsf{L}(1,1) = \sqrt{\mathsf{B}(1,1)}$, $\mathsf{L}(2:n,1) = \mathsf{B}(2:n,1)/\mathsf{L}(1,1)$, and $\mathsf{L}(2:n,2:n)$ is the Cholesky factor of $\mathsf{B}(2:n,2:n) - \mathsf{L}(2:n,1)\cdot\mathsf{L}(2:n,1)^\top$. Resolving this recursion into a loop leads to Algorithm 6, in which the lower triangle of the matrix B is overwritten with the Cholesky factor L.

By replacing matrix entries $\mathsf{B}(i,j)$ with n_b-by-n_b blocks $\mathsf{B}[i,j] := \mathsf{B}((i-1)n_b+1 : in_b, (j-1)n_b+1 : jn_b)$ we arrive at Algorithm 7 ($N = \lceil n/n_b\rceil$ is the number of blocks).

Both algorithms require roughly $\frac{1}{3}n^3$ flop. The operations of Algorithm 7 are done almost exclusively with level 3 BLAS.

Algorithm 6 Cholesky decomposition.

for $k = 1$ **to** n

$$\begin{aligned}
&\mathsf{B}(k,k) := \sqrt{\mathsf{B}(k,k)} \\
&\mathsf{B}(k+1:n,k) := \mathsf{B}(k+1:n,k) \cdot \frac{1}{\mathsf{B}(k,k)} \\
&\mathsf{B}(k+1:n,k+1:n) := \mathsf{B}(k+1:n,k+1:n) \\
&\qquad\qquad - \mathsf{B}(k+1:n,k) \cdot \mathsf{B}(k+1:n,k)^\top
\end{aligned}$$

Algorithm 7 Blocked Cholesky decomposition.

for $k = 1$ **to** N

$$\begin{aligned}
&\mathsf{L}[k,k] := \text{Cholesky factor of } \mathsf{B}[k,k] \qquad \{ \text{ Algorithm 6, overwriting } \mathsf{B}[k,k] \ \} \\
&\mathsf{B}[k+1:N,k] := \mathsf{B}[k+1:N,k] \cdot \mathsf{L}[k,k]^{-\top} \\
&\mathsf{B}[k+1:N,k+1:N] := \mathsf{B}[k+1:N,k+1:N] \\
&\qquad\qquad - \mathsf{B}[k+1:N,k] \cdot \mathsf{B}[k+1:N,k]^\top
\end{aligned}$$

5 Phase II: Methods for Tridiagonal Matrices

Once the symmetric matrix A is reduced to tridiagonal form, the eigenvalues λ_i and the eigenvectors $\mathbf{v}_i$ of the tridiagonal matrix T must be found. There is a large variety of algorithms for solving this problem.

Since its invention in the 1960's, the QR iteration has been the the method of choice for computing *all* eigenvalues (and, optionally, all eigenvectors). If only *selected eigenvalues* are required then bisection is the adequate method, otherwise QR-style algorithms tend to be faster. Inverse iteration may be used to compute *selected eigenvectors*, the subset being chosen after having determined and inspected the eigenvalues. In the 1990's the new divide-and-conquer algorithm, which has been developed with the aim of exposing parallelism, has proved by far superior to the QR iteration even on serial computers. It requires, however, significantly more memory than QR. Finally, there are other techniques — like homotopy algorithms — that are still in an experimental state.

Thoughout this section, we will use the shorthands $\mathsf{T}(i,i) =: \alpha_i$ and $\mathsf{T}(i,i-1) =: \beta_i$ for the diagonal and subdiagonal entries, respectively, of the symmetric tridiagonal matrix T, so

$$\mathsf{T} = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}.$$

We assume that all β_i are nonzero because otherwise the problem splits into two smaller subproblems that can be handled independently, cf. Section 2.5.

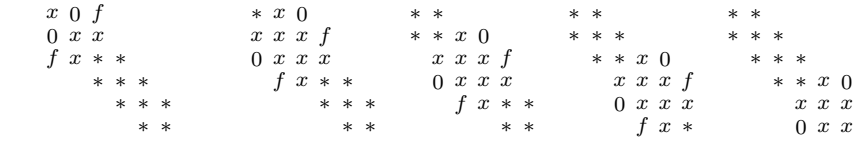

Figure 3. Rotations in one sweep of the QR iteration ($n = 6$).

5.1 QR Iteration

The QR iteration[17] for symmetric tridiagonal matrices is summarized in Algorithm 8. One pass through the **repeat** loop is called a *sweep* of the iteration.

Algorithm 8 QR iteration for tridiagonal matrices.

repeat
  determine a suitable shift $\sigma \in \mathbb{R}$
  apply a rotation in the $(1, 2)$ plane that zeroes the second entry of the vector $(\alpha_1 - \sigma, \beta_2, 0, \ldots, 0)^\top$. This creates a fill-in element at position $(3, 1)$.
  for $i = 2$ **to** $n - 1$
    apply a rotation in the $(i, i + 1)$ plane that zeroes the fill-in element at position $(i + 1, i - 1)$ and creates a new fill-in element at $(i + 2, i)$
until some subdiagonal entry β_i becomes negligible
set $\beta_i := 0$ and apply the whole algorithm to the submatrices $\mathsf{T}(1 : i, 1 : i)$ and $\mathsf{T}(i + 1 : n, i + 1 : n)$ { deflation }

Each sweep is initiated with a rotation $\mathsf{T} \mapsto \mathsf{R}^\top \mathsf{T} \mathsf{R}$, where the rotation angle depends on a certain parameter σ (the *shift*). This rotation produces a fill-in element below the subdiagonal, and a whole sequence of additional rotations is used to chase the fill element down the band until the tridiagonal structure is restored, cf. Figure 3. (See also Section 4.4 for a similar chasing strategy.)

If the shifts are chosen appropriately then the subdiagonal entries β_i tend to zero. Typically, the last subdiagonal entry β_n is the first to become negligible so that the last line in the algorithm reduces to continuing with the matrix $\mathsf{T}(1 : n - 1, 1 : n - 1)$, and that one eigenvalue may be read off from the diagonal entry α_n. But it is also possible that the matrix splits somewhere in the middle. The QR iteration tends to compute the eigenvalues by increasing absolute value, but this order may be broken.

The speed of convergence is determined by the shifts. For the most popular choice, *Wilkinson's shifts*, one can prove that the convergence is *global* (i.e., the subdiagonal elements are guaranteed to tend to zero) and — with rare exceptions — ultimately *cubic*. (Using Wilkinson's shifts means setting σ to one of the two eigenvalues of the current matrix T's trailing 2-by-2 subblock $\begin{pmatrix} \alpha_{n-1} & \beta_n \\ \beta_n & \alpha_n \end{pmatrix}$, namely

the eigenvalue closer to α_n.) The high order of convergence explains why — after some initial "warming up" — only two sweeps (with order-of-n flop per sweep) are needed on average in order to split off another eigenvalue. Therefore only order-of-n^2 flop are required to compute all eigenvalues of T. If the eigenvectors of T are needed, too, then each rotation must also be applied to an n-by-n matrix V, $\mathsf{V} \mapsto \mathsf{VR}$, where V has been initialized as the orthogonal matrix Q_1 that reduced the full or banded matrix A to tridiagonal form. (In the full case Algorithm 3, initialized with $\mathsf{Q} := \mathsf{I}$, can be used to build the matrix Q_1 with $\frac{4}{3}n^3$ flop.) Accumulating V requires a total of roughly $6n^3$ flop. As the QR iteration relies completely on rotations it is backward stable.

The name of the method stems from the fact that one sweep of the iteration corresponds to first computing a *QR decomposition* $\mathsf{T} - \sigma\mathsf{I} =: \tilde{\mathsf{Q}}\mathsf{R}$ into an orthogonal matrix $\tilde{\mathsf{Q}}$ and an upper triangular matrix R, and then replacing T with $\mathsf{R}\tilde{\mathsf{Q}} + \sigma\mathsf{I}$.

This idea can also be applied to matrices with semibandwidth $b \rangle 1$ (indeed, even to full matrices), but for complexity reasons (one sweep then takes order-of-$b^2 n$ flop without the work on the eigenvectors) such matrices are typically reduced to tridiagonal form before the iteration is started. If many eigenvalues but only a few eigenvectors of the banded matrix A are required then the following hybrid technique is may pay: First reduce A to tridiagonal form *without* accumulating the transformations, then compute the eigenvalues of T, and finally use the computed eigenvalues as "perfect" shifts in the QR iteration on the original A (with accumulation of the transformations in a matrix V, initialized as $\mathsf{V} = \mathsf{I}$) in order to obtain the required eigenvectors.

There is a variety of mostly newer (and often more efficient) methods that are in some way similar to the tridiagonal QR iteration, in particular the LR iteration, which relies on LR (i.e., LU) decompositions of the matrix, the QL iteration, which works bottom-up instead of top-down, the Pal-Walker-Kahan QR variant for computing eigenvalues only, which requires no squares roots, and the new qd algorithms[33], which essentially perform LR iteration on a factored representation of the tridiagonal matrix.

All these variants bear very limited potential for parallelism in the work on T. By contrast, the accumulation of the eigenvector matrix V is easily parallelized.

5.2 Bisection

Bisection is a versatile method for computing all or selected eigenvalues of T. It is based on the fact that for any $\mu \in \mathbb{R}$, the number $\nu(\mu)$ of negative elements in the sequence

$$\begin{aligned} \delta_1 &:= \alpha_1 - \mu \ , \\ \delta_i &:= (\alpha_i - \mu) - \frac{\beta_i^2}{\delta_{i-1}} \ , \quad i = 2, \ldots, n \ , \end{aligned} \tag{21}$$

is equal to the number of eigenvalues of T that are smaller than μ. (This is just Sylvester's law of inertia[34], applied to the decomposition $\mathsf{T} - \mu\mathsf{I} = \mathsf{L}\mathsf{D}\mathsf{L}^\top$ with a lower triangular matrix L having all ones on the diagonal, and a diagonal matrix D.)

Therefore, given an interval $[a, b)$ that is known to include the k-smallest eigenvalue λ_k of T, we may locate this eigenvalue to very high precision by repeatedly cutting the interval into two halves and testing which of the halves contains λ_k, see Algorithm 9. Note that a suitable initial "search interval" is provided by Gershgorin's theorem, which states that *all* eigenvalues of T are contained in the interval

$$I = [\ \min\{\alpha_i - \rho_i\ :\ i = 1, \ldots, n\}\ ,\ \ \max\{\alpha_i + \rho_i\ :\ i = 1, \ldots, n\}\ ]\ ,$$

where $\rho_i = |\beta_i| + |\beta_{i+1}|$, and $\beta_1 := 0$ and $\beta_{n+1} := 0$ for convenience.

Algorithm 9 Bisection for locating eigenvalue λ_k in an interval $[a, b)$.

compute $\nu(a)$ and $\nu(b)$ by building the sequences (21) for $\mu = a$ and $\mu = b$
if $\nu(a) \langle k \leq \nu(b)$ { otherwise λ_k is not contained in $[a, b]$ }
  while the interval width $b - a$ is too large
    let $c = (a + b)/2$ and compute $\nu(c)$ by evaluating (21) for $\mu = c$
    if $\nu(c) \geq k$ { λ_k lies in the left half of $[a, b]$ }
      $b := c$
    else { λ_k lies in the right half of $[a, b]$ }
      $a := c$
  return $\lambda_k \approx (a + b)/2$

Algorithm 9 is easily modified to compute a sequence $\lambda_j, \ldots, \lambda_k$ of consecutive eigenvalues, or all eigenvalues in a given interval $[a, b)$. Although bisection does not rely on orthogonal transformations (indeed, it is related to the unstable Gaussian elimination *without pivoting*), it is a perfectly stable algorithm featuring a very low backward error. In addition, bisection is efficiently parallelizable by having different processors compute disjoint subsets of the desired eigenvalues.

The major drawback of the method is its slow (namely, linear) convergence. This problem can be alleviated to some extent by using bisection only for *isolating* the eigenvalues, i.e., for narrowing down the initial interval until each subinterval contains exactly one eigenvalue. Then we switch to a superlinearly convergent root-finder, like Newton's method[25] or `zeroin`[8], for obtaining these eigenvalues to higher accuracy. Nevertheless, QR-type methods are superior on serial machines if more than one third, say, of the spectrum is required.

5.3 Inverse Iteration

Inverse iteration complements bisection in that it allows determining eigenvectors to selected previously computed eigenvalues. Inverse iteration is based on the *power iteration*, which is shown in Algorithm 10. The normalization (last line of the algorithm) is necessary to avoid overflow.

If there is a simple *dominant eigenvalue* (i.e., $|\lambda_{\max}| \rangle |\lambda_i|$ for the $n - 1$ other eigenvalues) then the vectors $\mathrm{sign}(\lambda_{\max}) \cdot \mathbf{v}$ tend to an eigenvector corresponding to $\lambda_{\max}$ (the *dominant eigenvector*), and $\|\tilde{\mathbf{v}}\|$ approaches $|\lambda_{\max}|$. The convergence is linear with factor $c = |\lambda_{\max}/\lambda_{\max_2}|$, where $\lambda_{\max_2}$ is the eigenvalue with second largest modulus. Therefore, the more dominant $\lambda_{\max}$ is, the faster the convergence.

Now, if $(\lambda_i, \mathbf{v}_i)$ is an eigenpair of T, and $\mu \in \mathbb{R}$, then $(\lambda_i - \mu, \mathbf{v}_i)$ is an eigenpair of $\mathsf{T} - \mu\mathsf{I}$, and $(1/(\lambda_i - \mu), \mathbf{v}_i)$ is an eigenpair of $(\mathsf{T} - \mu\mathsf{I})^{-1}$. Therefore, if μ is a very

Algorithm 10 Power iteration for approximating the dominant eigenvector.

select a suitable starting vector $\mathbf{v}$
repeat
    $\tilde{\mathbf{v}} := \mathsf{T} \cdot \mathbf{v}$
    $\mathbf{v} := \tilde{\mathbf{v}}/\|\tilde{\mathbf{v}}\|$
until $\mathbf{v}$ is "good enough"

good approximation to λ_i (i.e., $|\lambda_i - \mu| \ll |\lambda_j - \mu|$ for all $j \neq i$) then $1/(\lambda_i - \mu)$ is a strongly dominant eigenvalue of $(\mathsf{T} - \mu\mathsf{I})^{-1}$. This is the basis of inverse iteration, which is just the power iteration applied to $(\mathsf{T} - \mu\mathsf{I})^{-1}$ and therefore can be obtained by substituting "solve the system $(\mathsf{T} - \mu\mathsf{I}) \cdot \tilde{\mathbf{v}} = \mathbf{v}$" for the third line in Algorithm 10.

Under favorable circumstances inverse iteration is a very flexible and fast method, able to compute just the needed eigenvectors and requiring only order-of-n flop per eigenvector, one order of magnitude less than QR iteration! This is the case if the eigenvalues of T are well separated and if good approximations to them have been computed (e.g., with bisection or QR iteration). Then experience shows that for each eigenvalue *only one or two* iteration steps are needed to obtain an excellent approximation to the corresponding eigenvector. In addition, eigenvectors to different eigenvalues can be computed in parallel.

Problems occur with *clustered* eigenvalues, i.e., $\lambda_i \approx \lambda_{i+1} \approx \ldots \approx \lambda_j$. Then the orthogonality of the corresponding computed eigenvectors $\mathsf{fl}(\mathbf{v}_i), \ldots, \mathsf{fl}(\mathbf{v}_j)$ gets impaired because the clustered eigenvalues lead to almost identical linear systems $(\mathsf{T} - \mu\mathsf{I})\tilde{\mathbf{v}} = \mathbf{v}$, which in turn lead to similar solutions $\tilde{\mathbf{v}}$. This is particularly true if the starting vector $\mathbf{v}$ always remains the same. Therefore, one partial remedy is to use a new random starting vector for each eigenvalue. In addition, the vectors are explicitly *orthogonalized* against each other. Suppose that we already have computed mutually orthogonal, normalized eigenvectors for the eigenvalues $\lambda_i, \ldots, \lambda_\ell$ of the cluster. Then, in each iteration step for eigenvector $\mathbf{v}_{\ell+1}$, we apply the *modified Gram-Schmidt process* to the intermediate vector $\tilde{\mathbf{v}}$ before it is normalized. That is, for $r = i, \ldots, \ell$ the component of $\tilde{\mathbf{v}}$ in direction $\mathbf{v}_r$ is eliminated by replacing $\tilde{\mathbf{v}}$ with $\tilde{\mathbf{v}} - (\mathbf{v}_r^\top \tilde{\mathbf{v}}) \cdot \mathbf{v}_r$. Note that the explicit orthogonalization may sum up to order-of-n^3 flop if large clusters of eigenvalues are present. Both cures cannot preclude another type of failure, which amounts to matching the eigenvectors with wrong eigenvalues[23].

Currently a new variant of inverse iteration is being developed that always gives orthonormal eigenvectors *without* explicit orthogonalization[12]. If these efforts are successful then the combination of bisection (or QR iteration) with the new method will make all other tridiagonal eigensolvers obsolete.

5.4 Divide-and-Conquer

The divide-and-conquer algorithm[10,16] was developed with the aim of exploiting parallelism. As it turned out, this method typically beats the QR iteration and bisection/inverse iteration even on serial machines.

The first step in the divide-and-conquer algorithm consists of "tearing" the tridiagonal matrix into two halves via a suitable rank-1-modification. For a 6-by-6 matrix, this tearing might look like

$$\mathsf{T} = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \beta_4 & \alpha_4 & \beta_5 & \\ & & & \beta_5 & \alpha_5 & \beta_6 \\ & & & & \beta_6 & \alpha_6 \end{pmatrix} = \left(\begin{array}{ccc|ccc} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 - \beta_4 & & & \\ \hline & & & \alpha_4 - \beta_4 & \beta_5 & \\ & & & \beta_5 & \alpha_5 & \beta_6 \\ & & & & \beta_6 & \alpha_6 \end{array}\right) + \begin{pmatrix} & & & & & \\ & & & & & \\ & & & \beta_4 & \beta_4 & \\ & & & \beta_4 & \beta_4 & \\ & & & & & \\ & & & & & \end{pmatrix}$$

$$=: \left(\begin{array}{c|c} \mathsf{T}_1 & \\ \hline & \mathsf{T}_2 \end{array}\right) + \rho \mathbf{w}\mathbf{w}^\top$$

with $\rho = \beta_4$ and $\mathbf{w} = (0,0,1,1,0,0)^\top$. In general, T_1 and T_2 are m-by-m and $(n-m)$-by-$(n-m)$, resp., where $m \approx n/2$, $\rho = \beta_{m+1}$, and $\mathbf{w} = (0,\ldots,0,1,1,0,\ldots,0)^\top$.

Then the eigendecompositions $\mathsf{T}_1 = \mathsf{X}_1 \Delta_1 \mathsf{X}_1^\top$ and $\mathsf{T}_2 = \mathsf{X}_2 \Delta_2 \mathsf{X}_2^\top$ are computed by applying the divide-and-conquer algorithm recursively to the smaller matrices T_i (or, if these are small enough, QR iteration is used instead). This yields

$$\begin{aligned} \mathsf{T} &= \begin{pmatrix} \mathsf{X}_1 \Delta_1 \mathsf{X}_1^\top & \\ & \mathsf{X}_2 \Delta_2 \mathsf{X}_2^\top \end{pmatrix} + \rho \mathbf{w}\mathbf{w}^\top \\ &= \begin{pmatrix} \mathsf{X}_1 & \\ & \mathsf{X}_2 \end{pmatrix} \cdot \left(\begin{pmatrix} \Delta_1 & \\ & \Delta_2 \end{pmatrix} + \rho \mathbf{z}\mathbf{z}^\top \right) \cdot \begin{pmatrix} \mathsf{X}_1^\top & \\ & \mathsf{X}_2^\top \end{pmatrix} \\ &=: \mathsf{X} \cdot (\Delta + \rho \mathbf{z}\mathbf{z}^\top) \cdot \mathsf{X}^\top \ , \end{aligned}$$

where

$$\mathbf{z} = \mathsf{X}^\top \cdot \mathbf{w} = \begin{pmatrix} \text{last column of } \mathsf{X}_1^\top \\ \text{first column of } \mathsf{X}_2^\top \end{pmatrix} \ .$$

Then the eigendecomposition

$$\Delta + \rho \mathbf{z}\mathbf{z}^\top = \mathsf{Y} \Lambda \mathsf{Y}^\top \tag{22}$$

of a rank-1-perturbed diagonal matrix must be computed (see below), and finally the eigendecomposition of T can be recovered via $\mathsf{T} = (\mathsf{X}\mathsf{Y}) \cdot \Lambda \cdot (\mathsf{X}\mathsf{Y})^\top$.

The crucial step in the overall algorithm is the efficient and stable computation of the eigendecomposition (22), the remaining operations being expensive but trivial.

Suppose for a moment that all the entries of $\mathbf{z} = (\zeta_1, \ldots, \zeta_n)^\top$ are nonzero and that the eigenvalues (i.e., the diagonal entries) of Δ are distinct: $\delta_1 \langle \delta_2 \langle \ldots \langle \delta_n$. Then it is easy to show that the eigenvalues λ_i of $\Delta + \rho \mathbf{z}\mathbf{z}^\top$ are the roots of the so-called *secular equation*

$$f(\lambda) := 1 + \rho \sum_{i=1}^{n} \frac{\zeta_i^2}{\delta_i - \lambda} = 0 \ , \tag{23}$$

and that for any eigenvalue λ_i of $\Delta + \rho \mathbf{z}\mathbf{z}^\top$,

$$\mathbf{y}_i := (\Delta - \lambda_i \mathsf{I})^{-1} \cdot \mathbf{z} \tag{24}$$

is a corresponding eigenvector. (Note that computing $\mathbf{y}_i$ involves just an appropriate entry-wise scaling of $\mathbf{z}$.)

Eq. (23) implies that the δ_i "interlace" the sought eigenvalues, i.e., for $\rho\rangle 0$ we have

$$\delta_1\langle\lambda_1\langle\delta_2\langle\lambda_2\langle\delta_3\langle\ldots\langle\delta_n\langle\lambda_n \ .$$

This property, together with the fact that on each interval (δ_i, δ_{i+1}) the function f can be approximated by simple rational expressions, leads to a globally and quadratically convergent root finder that can compute all the λ_i in order-of-n^2 time.

Unfortunately, Eq. (24) is not an adequate means to compute the eigenvectors because orthogonality is severely impaired in the presence of close δ_i. It took more than ten years from the invention of the divide-and-conquer method until a technique was discovered[19] that did not need resorting to extended precision in the eigenvector computations. Roughly speaking, we can compute orthogonal eigenvectors for $\mathsf{\Delta} + \rho\mathbf{z}\mathbf{z}^\top$ by applying Formula (24) to a slightly modified vector $\mathbf{z}$.

In practice our assumption that all the ζ_i are nonzero and all the δ_i are distinct is seldom fulfilled. The superiority of the divide-and-conquer method comes from the fact that it even can take advantage from a violation of this assumption. In fact, if some ζ_i is zero then δ_i and the ith column of X already are an eigenpair of the tridiagonal matrix T. Similarly, if some of the δ_i are (almost) identical then all but one of them are also very good approximations to eigenvalues λ_i of T, and the corresponding eigenvalues can be computed cheaply. Thus, the eigenvalue problem for $\mathsf{\Delta} + \rho\mathbf{z}\mathbf{z}^\top$ and the ensuing multiplication $\mathsf{X} \cdot \mathsf{Y}$ are effectively reduced in size. Fortunately this type of *deflation* is quite frequent.

Parallelism can be exploited in two ways. First, all the solutions of the secular equation and the corresponding eigenvectors of $\mathsf{\Delta} + \rho\mathbf{z}\mathbf{z}^\top$ may be computed independently from each other, and second, the matrix–matrix product $\mathsf{X}\cdot\mathsf{Y}$ lends itself naturally to a coarse-grained parallelization.

There are also attempts to apply the divide-and-conquer technique to banded matrices[3], but the resulting algorithms are still highly experimental.

5.5 Homotopy Methods

Homotopy methods[29] try to follow the paths of the eigenvalues and eigenvectors through a whole sequence of matrices

$$\mathsf{T}_0 \ \mapsto \ \mathsf{T}_1 = \mathsf{T}_0 + \theta_1(\mathsf{T} - \mathsf{T}_0) \ \mapsto \ \ldots \ \mapsto \ \mathsf{T}_{k-1} = \mathsf{T}_0 + \theta_{k-1}(\mathsf{T} - \mathsf{T}_0) \ \mapsto \ \mathsf{T}_k = \mathsf{T} \ ,$$

where T_0 is some initial matrix whose eigensystem is readily computed (e.g., a diagonal matrix), T is the matrix whose eigensystem is sought, and $0 = \theta_0\langle\theta_1\langle\ldots\langle\theta_{k-1}\langle\theta_k = 1$. If the step-sizes $\theta_i - \theta_{i-1}$ are small enough then the eigenvalues of T_{i-1} are good starting values for computing the eigenvalues of T_i, and inverse iteration for T_i's eigenvectors can be started with the eigenvectors of T_{i-1}.

The same idea may also be applied to other (e.g., banded) matrices and to the generalized eigenvalue problem. Note that the homotopy methods are still in an experimental state.

6 Methods Without Initial Tridiagonalization

In contrast to the methods described above, the algorithms discussed in this section do not rely on an initial tridiagonalization of the matrix, but apply the iterative process to the matrix A itself.

6.1 Jacobi's Method

Jacobi's method is based on the idea of reducing A's "off-diagonal norm",

$$\mathrm{off}(\mathsf{A}) := \sqrt{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \mathsf{A}(i,j)^2} \ ,$$

until it is negligibly small. Then we can read off A's eigenvalue from the diagonal entries and the eigenvectors from the columns of the orthogonal transformation matrix that was used to attain the almost-diagonal form.

This form is achieved by repeatedly zeroing selected entries $\mathsf{A}(i,j)$ with suitable rotations. If we choose the rotation angle θ such that

$$\tan\theta = \frac{\mathrm{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}} \ , \qquad \text{where} \quad \tau = \frac{\mathsf{A}(i,i) - \mathsf{A}(j,j)}{2\mathsf{A}(i,j)} \ ,$$

then a short computations shows that the two-sided rotation $\mathsf{A} \mapsto \mathsf{R}^\top \mathsf{A} \mathsf{R}$ with $\mathsf{R} = \mathsf{R}(i,j,\theta)$ indeed zeroes A's (i,j) and (j,i) entries and that $\mathrm{off}(\mathsf{A})$ drops by $2 \cdot \mathsf{A}(i,j)^2$. Note that again the rotation angle is not needed explicitly because the parameters c and s may be obtained via

$$c = \frac{1}{\sqrt{1+t^2}} \ , \quad s = t \cdot c \ .$$

Unfortunately, zeros introduced this way do not persist but are made nonzero again in later rotations. Thus most entries of A have to be made zero several times during the whole process. There are many different strategies for selecting the order of the entries (i,j) to be zeroed.

Obviously, zeroing the off-diagonal entry $\mathsf{A}(i,j)$ with the largest absolute value will lead to the largest reduction of $\mathrm{off}(\mathsf{A})$. This is the *classical Jacobi method*[22], which is slowed down by organizational overhead since the roughly $n^2/2$ comparisons for determining the maximum entry cost by far more time than the $6n$ ensuing arithmetic operations.

Therefore most often cheaper schemes are used, in particular the *row cyclic* and *column cyclic* elimination orders. In the former, the entries of the strictly lower triangle of A are made zero row-by-row, that is, in the order $\mathsf{A}(2,1)$, $\mathsf{A}(3,1)$, $\mathsf{A}(3,2)$, $\mathsf{A}(4,1)$, $\mathsf{A}(4,2)$, $\mathsf{A}(4,3)$, $\ldots$, $\mathsf{A}(n,1)$, $\mathsf{A}(n,2)$, $\ldots$, $\mathsf{A}(n,n-1)$. When all these entries have been made zero once (this is called a *sweep* of the method), then the process is started anew.

It is easy to see that each rotation in the classical method must reduce $\mathrm{off}(\mathsf{A})$ by a factor $\leq 1 - \frac{2}{n(n-1)} \langle 1$, thus implying at least linear convergence. A more involved analysis reveals that the convergence is indeed much faster, namely quadratic[34].

This is also true for the cyclic schemes, provided that some precautions concerning the rotation angles are taken.

Experimental evidence suggests that in practice roughly $\log n$ sweeps are necessary to achieve adequate accuracy. As the methods discussed in Sections 4 and 5 require much less work (corresponding to just two Jacobi sweeps), Jacobi's method is usually not competitive. If, however, the matrix A is already strongly *diagonally dominant* (i.e., its diagonal entries are much larger than the off-diagonals) then Jacobi's method needs only a few sweeps to converge and may even beat the reduction-based methods.

Two other facts have revived the interest in Jacobi's method. First, it was observed that in some cases this algorithm — carefully implemented — can deliver much more accurate eigensystems than the other techniques. And second, appropriate cyclic elimination schemes allow exploiting parallelism by applying several rotations simultaneously[31].

There are also variants of Jacobi's method for the generalized eigenvalue problem, but their convergence properties are not sufficiently known.

6.2 Invariant Subspace Decomposition

The *invariant subspace decomposition algorithm*[21] (ISDA) is in some sense dual to the bisection/inverse iteration approach since it extracts information about the eigenvectors *before* the eigenvalues. This algorithm relies on the fact that eigenvectors are invariant under polynomial transformations of the matrix: If $(\lambda, \mathbf{q})$ is an eigenpair of A then for any polynomial p, $(p(\lambda), \mathbf{q})$ is an eigenpair of $p(\mathsf{A})$.

First, lower and upper bounds for $\mathrm{spec}(\mathsf{A})$ are determined (e.g., with Gershgorin's theorem for full matrices), and then a linear transformation $\mathsf{A} \mapsto \alpha\mathsf{A} + \beta\mathsf{I} =: \mathsf{A}_1$ is applied that maps the lower half of the eigenvalues, $\lambda_1, \ldots, \lambda_{n/2}$, into the interval $[0, \frac{1}{2}]$ and the upper half of the eigenvalues into $[\frac{1}{2}, 1]$.

Then further polynomial transformations $\mathsf{A}_k \mapsto p_k(\mathsf{A}_k) =: \mathsf{A}_{k+1}$ are applied, where the polynomial p_k is designed such that $p_k(x) \approx 0$ for $x \in [0, \frac{1}{2})$ and $p_k(x) \approx 1$ for $x \in (\frac{1}{2}, 1]$, i.e., p_k pushes the lower half of the eigenvalues toward 0 and the upper half toward 1. This process is repeated until $\mathsf{A}_k \approx \mathsf{A}_{k-1}$, implying that all eigenvalues of the final A_k are approximately 0 or 1.

Next, a so-called *rank-revealing QR decomposition* $\mathsf{A}_k = \mathsf{QR}$ of this matrix into an orthogonal matrix Q and an upper triangular matrix R is determined. Then the first (last) $n/2$ columns of Q are orthonormal eigenvectors to the eigenvalue 1 (0, respectively) of A_k. This implies

$$\mathsf{Q}^\top \mathsf{A} \mathsf{Q} = \begin{pmatrix} \mathsf{A}' & \\ & \mathsf{A}'' \end{pmatrix} ,$$

where A' and A'' are symmetric $(n/2)$-by-$(n/2)$ matrices. According to Section 2.5, the eigenvalues and eigenvectors of A may now be obtained from the eigendecompositions of A' and A'', which in turn are computed by recursively applying the ISDA to these two matrices, or with QR iteration if A' and A'' are small enough.

This approach does most of its computations with matrix–matrix products (e.g., in the evaluation of $p_k(\mathsf{A}_k)$) and therefore achieves high $\mathsf{Mflop/s}$ rates. In addition,

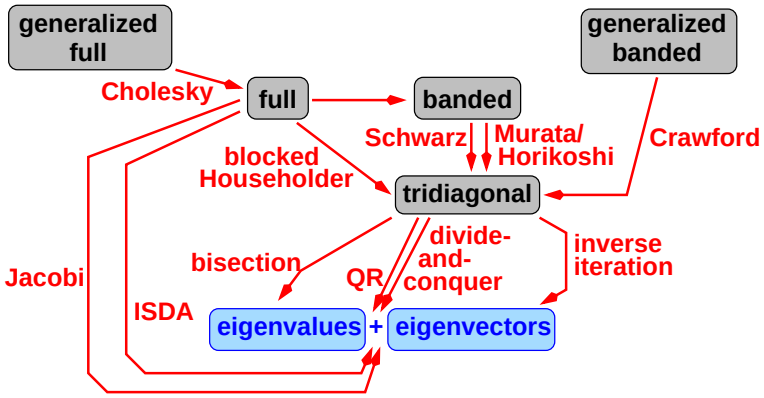

Figure 4. Computational paths for the symmetric eigenvalue problem.

efficient parallelization is possible. On the other hand, the ISDA requires several times more flop than the reduction–based techniques and is therefore not competitive on serial machines.

7 Synopsis

Figure 4 summarizes the algorithms used in direct symmetric eigensolvers. Depending on the initial problem (standard or generalized, full or banded), on the required information (eigenvalues only or eigenvectors, too), and on a priori knowledge (are the eigenvalues clustered or not?), different computational paths through the diagram are taken.

8 Available Software

Whenever possible, eigenvalue solvers from established libraries should be used because much effort went into optimizing their performance and making them robust (e.g., appropriate scalings for "balancing" the eigenvalues). There are many pitfalls awaiting the ambitious but unexperienced programmer.

8.1 Serial and Shared-Memory Machines

As modern software for dense or banded matrices relies heavily on the BLAS, obtaining an optimized implementation of the BLAS is extremely important. For most high-performance machines, optimized BLAS are provided by the manufacturer. If this is not the case, the public-domain BLAS from the ATLAS project (`http://www.netlib.org/atlas`) are a viable alternative. They often perform almost as well as (and sometimes even better than) proprietary implementations.

The public-domain *LAPACK* library[1] (`http://www.netlib.org/lapack`) contains optimized implementations for almost all non-experimental algorithms described in this article. (LAPACK also solves linear systems and least squares and

related problems, and most routines are available for real and complex matrices in either single-precision or double-precision arithmetic.) Preferably the libary's comfortable *driver routines* should be used. E.g., `DSYEV` computes the eigendecomposition of a full matrix, whereas `DSBEV` handles the banded case. Similar drivers exist for the generalized (full or banded) eigenvalue problem, as well as so-called "expert" drivers with additional options (e.g., computing only parts of the eigendecomposition). In addition, the computational routines may be called directly. Thus, `DSYTRD` (`DSBTRD`) reduces a full (banded) matrix to tridiagonal form with Algorithm 2 (with a modified version of Algorithm 4).

At `http://www-unix.mcs.anl.gov/prism`, additional software for the ISDA (Section 6.2) and for the two-phase reduction (mentioned at the end of Section 4.2) is available.

Because of the high quality of the LAPACK library, many vendor-supplied numerical packages and commercial libraries (like NAG) are based on these routines. This is also the case for *multithreaded* parallel libraries that come with shared-memory parallel machines. Here the parallelism is often confined within the BLAS.

8.2 Distributed-Memory Systems

For distributed-memory machines with the message-passing programming model, the situation is more complicated. The direct analogue to LAPACK is the *ScaLAPACK* library[7] (`http://www.netlib.org/scalapack`), which contains the functionality of many LAPACK routines. ScaLAPACK is based on the *PBLAS*, a parallelized implementation of the BLAS. ScaLAPACK contains solvers for the symmetric standard and generalized eigenvalue problem with full matrices, but *no banded solvers*. Another problem with this library is the fact that inverse iteration performs explicit orthogonalization only against eigenvectors within the same processor. Thus excessive memory may be required on a single processor, or orthogonality may be lost if explicit orthogonalization is turned off. (ScaLAPACK also includes QR iteration, which does not suffer from this problem.)

PeIGS (`http://www.emsl.pnl.gov:2080/docs/global/peigs.html`) is another library offering eigensystem functionality comparable to that of ScaLAPACK; in particular, banded problems are not addressed. While ScaLAPACK relies on a two-dimensional data layout (i.e., the matrices are split along rows *and* columns), PeIGS works with matrices that are distributed by whole columns or whole rows. Such distributions are typically inferior. On the other hand, inverse iteration in PeIGS orthogonalizes against all eigenvectors of a cluster — also on different processors — and therefore gives better results.

Finally, there are packages that do not contain complete eigensolvers but rather provide infrastructure for easily putting such methods together, namely *PLAPACK*[38] (`http://www.cs.utexas.edu/users/plapack`) and *Global Arrays* `http://www.emsl.pnl.gov:2080/docs/global/ga.html`). Both packages facilitate the manipulation of distributed matrices.

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, PA (1999).
2. ANSI/IEEE, IEEE Standard for Binary Floating Point Arithmetic, Std. 754, New York (1985).
3. P. Arbenz, Divide and conquer algorithms for the bandsymmetric eigenvalue problem, *Parallel Comput.* **18**, 1105–1128 (1992).
4. J. Barlow and J. Demmel, Computing accurate eigensystems of scaled diagonally dominant matrices, *SIAM J. Numer. Anal.* **27(3)**, 762–791 (1990).
5. C. Bischof, B. Lang, and X. Sun, Parallel tridiagonalization through two-step band reduction, in *Proceedings of the Scalable High-Performance Computing Conference, Knoxville, Tennessee, May 23–25, 1994*, IEEE Computer Society, Los Alamitos, CA, 23–27 (1994).
6. C.H. Bischof and C.F. Van Loan, The WY representation for products of Householder matrices, *SIAM J. Sci. Stat. Comput.* **8 (1)**, s2–s13 (1987).
7. L.S. Blackford, J. Choi, A. Cleary, E. d'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker and R.R. Whaley, *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA (1997).
8. R.P. Brent, *Algorithms for Minimization Without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ (1973).
9. C.R. Crawford, Reduction of a band symmetric generalized eigenvalue problem, *Comm. Assoc. Comp. Mach.* **16**, 41–44 (1973).
10. J.J.M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numer. Math.* **36**, 177–195 (1981).
11. J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA (1997).
12. I.S. Dhillon, *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*, Ph.D. thesis, Computer Science Division, University of California at Berkeley (1997).
13. J.J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, A set of level 3 basic linear algebra subprograms, *ACM Trans. Math. Softw.* **16 (1)**, 1–17 (1990).
14. J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, An extended set of FORTRAN basic linear algebra subprograms, *ACM Trans. Math. Softw.* **14 (1)**, 1–17 (1988).
15. J.J. Dongarra, S.J. Hammarling, and D.C. Sorensen, Block reduction to condensed forms for eigenvalue computations, *J. Comput. Appl. Math.* **27**, 215–227 (1989).
16. J.J. Dongarra and D.C. Sorensen, A fully parallel algorithm for the symmetric eigenvalue problem, *SIAM J. Sci. Stat. Comput.* **8 (2)**, s139–s154 (1987).
17. J.G.F. Francis, The QR transformation: A unitary analogue to the LR transformation, part I and II, *Computer J.* **4**, 265–272 and 332–345 (1961/62).
18. G.H. Golub and C.F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD (1996).
19. M. Gu and S.C. Eisenstat, A stable and efficient algorithm for the rank-1

modification of the symmetric eigenvalue problem, *SIAM J. Matrix Anal. Appl.* **15**, 1266–1276 (1994).
20. N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA (1996).
21. S. Huss-Lederman, A. Tsao and G. Zhang, A parallel implementation of the invariant subspace decomposition algorithm for dense symmetric matrices, in *Proc. Sixth SIAM Conf. on Parallel Processing for Scientific Computing in Norfork, VA*, SIAM, Philadelphi, PA (1993).
22. C.G.J. Jacobi, Über ein leichtes Verfahren, die in der Theorie der Säculärstörungen vorkommenden Gleichungen zu lösen, *Crelle's J.* **30**, 51–94 (1846).
23. E.R. Jessup and I.C.F. Ipsen, Improving the accuracy of inverse iteration, *SIAM J. Sci. Stat. Comput.* **13**, 550–572 (1992).
24. L. Kaufman, Banded eigenvalue solvers on vector machines, *ACM Trans. Math. Softw.*, **10 (1)**, 73–86 (1984).
25. D. Kincaid and W. Cheney, *Numerical Analysis*, 2nd ed., Brooks/Cole Publishing Company, Pacific Grove, CA (1996).
26. B. Lang, A parallel algorithm for reducing symmetric banded matrices to tridiagonal form, *SIAM J. Sci. Comput.* **14 (6)**, 1320–1338 (1993).
27. B. Lang, Using level 3 BLAS in rotation-based algorithms, *SIAM J. Sci. Comput.* **19 (2)**, 626–634 (1998).
28. C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, Basic linear algebra subprograms for FORTRAN usage, *ACM Trans. Math. Softw.* **5 (3)**, 308–323 (1979).
29. T.Y. Li and N. Rhee, Homotopy algorithm for symmetric eigenvalue problems, *Numer. Math.* **55**, 265–280 (1989).
30. The MathWorks, Inc., *MATLAB Reference Guide*, Natick, MA (1999).
31. J.J. Modi, *Parallel Algorithms and Matrix Computation*, Clarendon Press, Oxford (1988).
32. K. Murata and K. Horikoshi, A new method for the tridiagonalization of the symmetric band matrix, *Information Processing in Japan* **15**, 108–112 (1975).
33. B.N. Parlett, The new qd algorithms, *Acta Numerica*, 459–491 (1995).
34. B.N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA (1998). Updated reprint of the 1980 Prentice-Hall edition.
35. W. Rath, Fast Givens rotations for orthogonal similarity transformations, *Numer. Math.* **40**, 47–56 (1982).
36. R. Schreiber and C. Van Loan, A storage efficient WY representation for products of Householder transformations, *SIAM J. Sci. Stat. Comput.* **10 (1)**, 53–57 (1989).
37. H.R. Schwarz, Tridiagonalization of a symmetric band matrix, *Numer. Math.* **12**, 231–241 (1968).
38. R.A. van de Geijn, *Using PLAPACK*, MIT Press, Cambridge (1997).
39. J. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall (1964).
40. J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford (1965).